Command Processing And Other Stuff

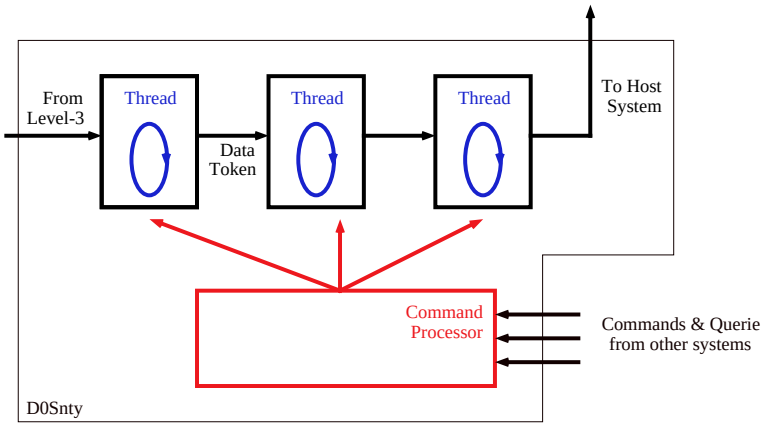
Gordon Watts
6-6-97
Online Meeting

- Command Processing
- Level-3 Monitor Data
- Errors
- Logs

You can find all of this talk on WWW:
http://d0sgi0.fnal.gov/ gwatts/talks/

Command Classes

First a little context (DØSNTY):



- Many threaded application
- Commands must interact with threads
- Commands aren't part of threads

Command Types

- Slow Commands
  - Wait for program to complete operation before disconnecting
  - Sometimes very long operations (seconds or minutes?).
  - Example: Reset/Restart, Empty Queues, etc.
- Quick Commands
  - Never requires interacting or blocking another thread
  - Start a operation but don't wait for its completion
  - Examples: Quit as soon as empty, How many events processed?

Slow Commands should not block Quick Commands

Other Requirements

- Command library is independent of I/O method
  - TCP/IP, DØIP, Named Pipes, etc.
- Independent of format of input data stream
  - text or binary
- Auto processing of incoming commands
- Commands can have conversations (context)
- Easy to use (Yeah, Right!).

Command Classes

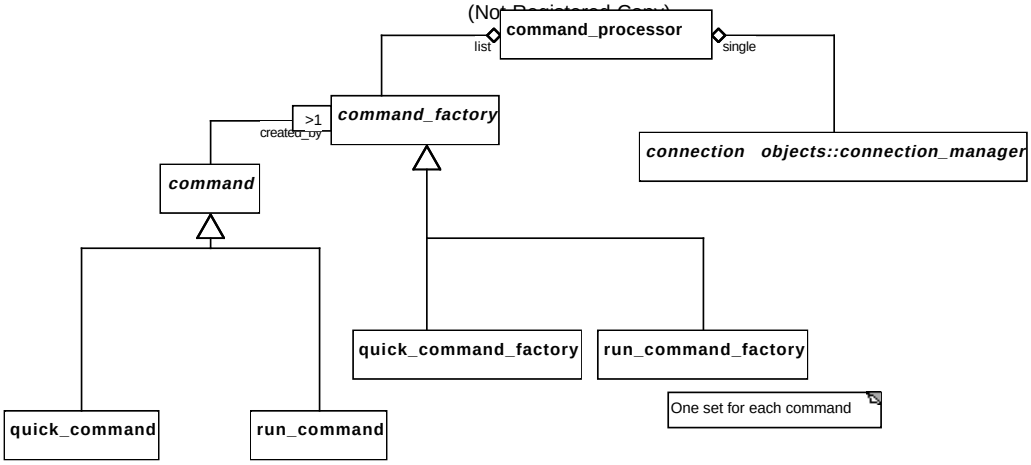
Everything based around the `command` class.

- Standard *execute* method
  - Takes a `connection_conversation` as an argument
- Scope is as long as command is active.

Command dispatch is done by the `command_processor` class.

- Maintains a list of `command_factory` classes
  - One `command_factory` per command
- Methods to manage list (add/remove `commands`).
- Knows about a `connecton_manager`

Command Classes



Connection Classes

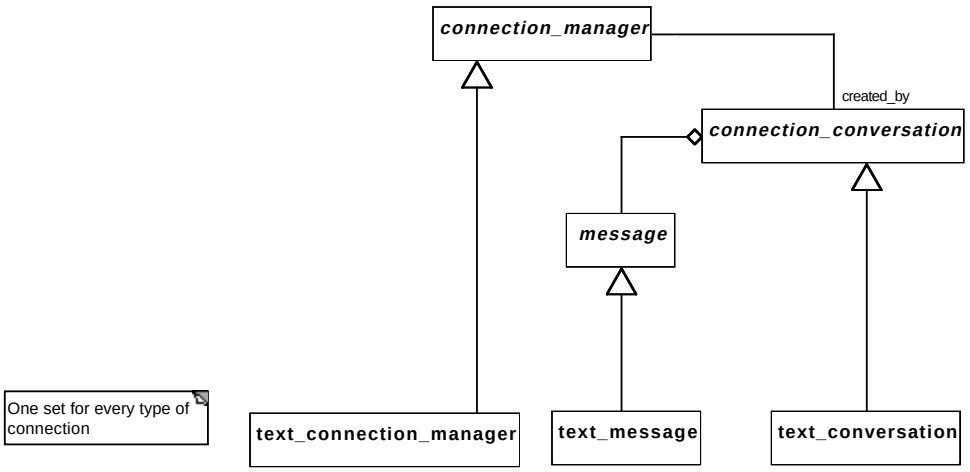
The `message` is the abstract object passed around

- Fed to and gotten from a `connection_conversation`
- Can block waiting for a new `message`

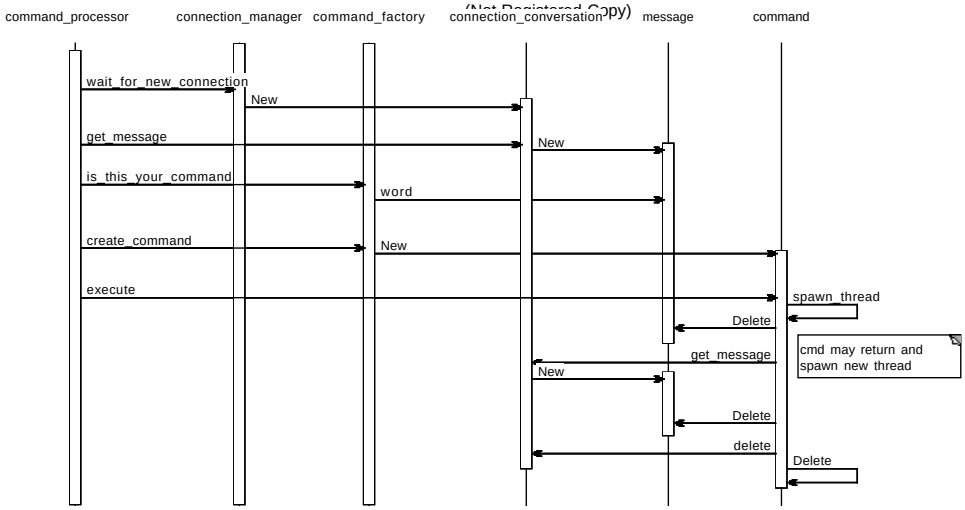
The `connection_manager` keeps track of connections.

- Waits for new connection to a *port*
- Blocks threads

Connection Classes



Event Trace



You Call This Simple!?

- Helper templates and classes for Text/Word based messages
- Build `command_factory` automatically
- Actual command process loop is tight and clean
- Must write the `command` subclass

Source Example

Define a new command:

```
class spit_cmd : public single_exchange_command {

public:

        inline spit_cmd (void) {};

        inline void execute_single_exchange
         (const connection_message _cmd_text)
            {std::cout << "this is junk" << std::endl;};
};
```

Main Program:

```
main ()
{
        std::cout << "Hi there" << endl;

        istream_connection_manager inputer (std::cin);
        command_processor cmd (inputer);

        cmd.add_command (new simple_command_factory<spit_cmd> ("spit"));

        cmd.process_single_command();

        std::cout << "Done" << endl;
}
```

What is there?

- All the command classes
- connection classes for `tty` input
- Next are classes for DØIP and NT Named Pipes
- And a harsher test program

Level-3 Monitor Data

Level-3 will have data that needs to be saved at the end of a run.

- Histograms (small), counters, perhaps some physics information.
- Could be something like ntuples of failed events (large data storage).
- Where should it go?
- How should it be handled?
- How big will it be?

Monitor Data Size

What if we wrote Ntuples on Failed Events?	
L2 Accept Rate (Hz)	1000
% L2 Evts Saved	10
LWords Per Event	4
Word/sec out of L3	400
KByte/sec out of L3	1.5625
Level3 Nodes	48
KBytes/sec/Node	0.032552083

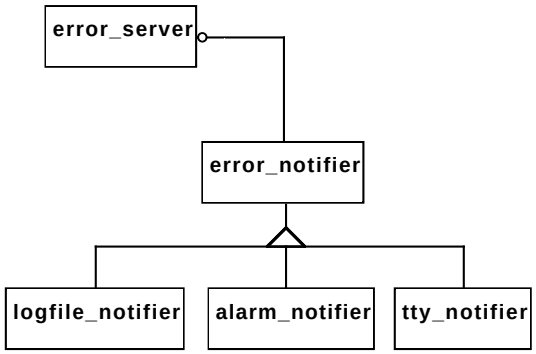
Data Storage Needs	
Accel Uptime (%)	50
Years Running	1
Mon Data (MBytes)	24060.0586
Mon Data (GBytes)	23.496151
# 27 GDisks/year	0.87022781

Error Reporting

From the Level-3 Point of View:

- Same online and offline API
- High level API simple (like last time).
- Low level allow plug in reporters
  - Text output plug in
  - TCP/IP output plugin
  - L3 interface plugin
- Too complex?

Error Classes?



Log Files

I have a fantasy... I have several actually...

- Get a call at 4am... Use browser to see what the DAQ shifter did
  - Really tough – all logs must be accessible by a HTTP server.
- Every three months burn a CD with all logs in HTML form
  - Indexed by Run
  - Indexed by Date

  and then delete them!

Log Files con't

Common Format?

- Each line contain a time stamp
- Logs contains run start/end entries
- Log files are periodically closed/new one started (by run, by time?).
- Makes for ease of indexing.

If we supply the software for this, it might be easy. A fighting chance.

Log Files con't

Auto Handling?

- Built into the API auto close/open
  - Time
  - Run
- Autonotificaton?
  - Log Manager Process
  - Copy log files to common area
  - Interleave log entries for huge list?
  - Do it on HTTP request?
- Is API same as for errors?

Conclusions

- This is the first time I've used OO design tools
- Good to start with
- Can't keep up once you start coding
- Are the `command` classes useful?
- Errors and Loggers. Get them early, we can have an integrated system
  - Just like the commands in the online system.